Data Structures
CSCI H343, Fall 2021

Midterm

Name: ____________________

This exam has 11 questions, for a total of 100 points.

1. [8 points] What is the output of this Java program?

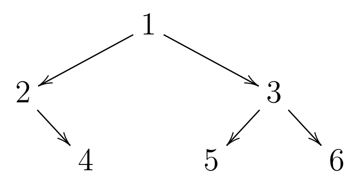
```
public class Node {
    Node left, right; int data;
    Node(int d) { data = d; }
    public void f() {
        System.out.print("(");
        if (left != null) {
            left.f();
            System.out.print("_");
        }
        if (right != null) {
            right.f();
            System.out.print("_");
        }
        System.out.print(data);
        System.out.print(")");
    }

    public static void main(String[] args) {
        Node a = new Node(0);
        Node b = new Node(1);
        Node c = new Node(2);
        Node d = new Node(3);
        a.left = b;
        a.right = c;
        c.left = d;
        a.f();
    }
}
```

Solution:

```
((1)_((3)_2)_0)
```

2. [8 points] Consider this binary tree.



The following questions are with respect to an inorder traversal.

1. Which node comes immediately after node 1?
2. Which node comes immediately after node 5?
3. Which node comes immediately before node 5?
4. Which node comes immediately before node 1?

Solution: 2 points each

1. 5
2. 3
3. 1
4. 4

3. [10 points] For the following `Node` class in a binary tree, Fill in the blanks to complete the following implementation of the `next` method that returns the node that comes after the current node with respect to an inorder traversal, if there is one, and `null` if there is none.

```
class Node {
    T data;
    Node left, right, parent;
    Node next() {
        if (right == null) {
            return ___(a)___;
        } else {
            return ___(b)___;
        }
    }
    Node first() {
        if (left == null) {
            return this;
        } else {
            return ___(c)__;
        }
    }
    Node nextAncestor() {
```

```
        Node n = this, p = parent;
        while (p != null && ___(d)___) {
            n = p;
            p = ___(e)___;
        }
        return p;
    }
}
```

Solution: Rubric: 2 points each

```
a) nextAncestor()
b) right.first()
c) left.first()
d) p.right == n
e) p.parent
```

4. [8 points] The `divide` function is meant to divide integer m by the integer n, returning the quotient q and remainder r. The integers m and n are required to be non-negative. The correctness criteria for `divide` is that the quotient q and remainder r should satisfy

$$m = nq + r \qquad 0 \leq r < n$$

```
Pair<Integer, Integer> divide(int m, int n) {
    int q = 0;
    int r = m;
    while (r >= n) {
        r = r - n;
        q = q + 1;
    }
    return new Pair<>(q, r);
}
```

1. State the loop invariant for the `while` loop.
2. Explain why the loop invariant is true before the start of the loop.
3. For a hypothetical iteration of the loop, explain why the loop invariant is true at the end of the loop body, assuming only that the loop invariant was true at the beginning of the loop body.
4. Explain why the loop invariant combined with the loop condition being false logically implies the correctness criteria for the `divide` function.

Solution:

1. The loop invariant is $m = nq+r$ and $0 \leq r$. **(2 points)** (OK if just $m = nq+r$.)
2. $m = n \cdot 0 + m$ **(2 points)** and from $0 \leq m$ we have $0 \leq r$.
3. We may assume to $m = nq+r$ at the beginning of the loop body. Let $r' = r-n$ and $q' = q+1$, so r' and q' are the values of `r` and `q` at the end of the loop body. We need to show that $m = nq' + r'$.

$$nq' + r' = n(q+1) + (r-n) = nq + n + r - n = nq + r = m$$

(2 points)
Also, from the loop $r \geq n$ we have $0 \leq r - n$.

4. The negation of $r \geq n$ is $r < n$, and the loop invariant is $m = nq+r$ and $0 \leq r$, which are identical to the correctness criteria for `divide`. **(2 points)**

5. [12 points] What is the big-O time complexity of the following `flood` method in terms of the total number of tiles, represented by n? Provide an argument for your answer that analyzes every statement in the method and how their individual time complexities combine into the total time complexity.

```
public static void flood(WaterColor color,
                         LinkedList<Coord> flooded_list,
                         Tile[][] tiles,
                         Integer board_size) {
    HashSet<Coord> flooded_set = new HashSet<>(flooded_list);
    ArrayList<Coord> flooded_array = new ArrayList<>(flooded_list);
    for (int i = 0; i != flooded_array.size(); ++i) {
        Coord c = flooded_array.get(i);
        for (Coord n : c.neighbors(board_size)) {
            if (!flooded_set.contains(n)
                  && tiles[n.getY()][n.getX()].getColor() == color) {
                flooded_array.add(n);
                flooded_list.add(n);
                flooded_set.add(n);
            }
        }
    }
}
```

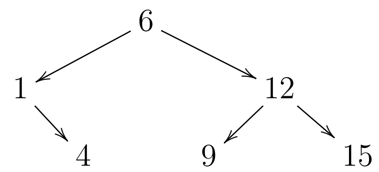
Solution:

The construction of the `flooded_set` and `flooded_array` are both $O(n)$. **(2 points)** Next we analyze the body of the outer `for` loop.

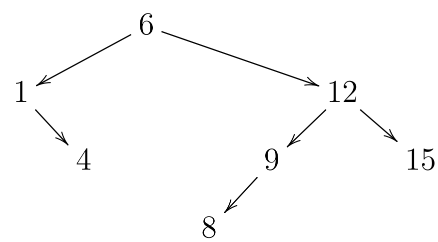
- The call `flooded_array.get(i)` is $O(1)$. **(2 points)**
- The inner `for` loop may iterate up to 4 times, so it doesn't matter **(1 point)**.
- The call to `flooded_set.contains(n)` is $O(1)$. **(2 points)**
- The `n.getX()`, `n.getY()`, `getColor()`, and the `add` methods are all $O(1)$ and the access to the `tiles` array is $O(1)$ **(1 point)**.

The maximum time complexity of the operations inside the outer `for` is $O(1)$, and there are at most n iterations because the `flooded_list` may contain up to n coordinates. so the time complexity of the outer `for` is $O(n)$ **(2 points)**. Adding this with the time for the construction of the `flooded_set` and `flooded_array` results in a time complexity of the `flood` method of $O(n)$ **(2 points)**.

6. [7 points] Draw the result of inserting key 8 into the following Binary Search Tree.



Solution:



7. 10 points Write the Java code for the implementation of the below `find_first_equal` function. Recall that the `Iterator` interfaces is defined as follows.

```
interface Iterator<T> {
     T get();
     void set(T e);
     void advance();
     void advance(int n);
     boolean equals(Iterator<T> other);
     Iterator<T> clone();
}
```

The `find_first_equal` function returns an iterator pointing to the first element in the half-open range `[begin,end)` that equals the 'x' parameter. If no element equals 'x', return the `end` iterator. The `begin` and `end` iterators must not be changed.

```
public static <E> Iterator<E>
find_first_equal(Iterator<E> begin, Iterator<E> end, E x) {



}
```

Solution:

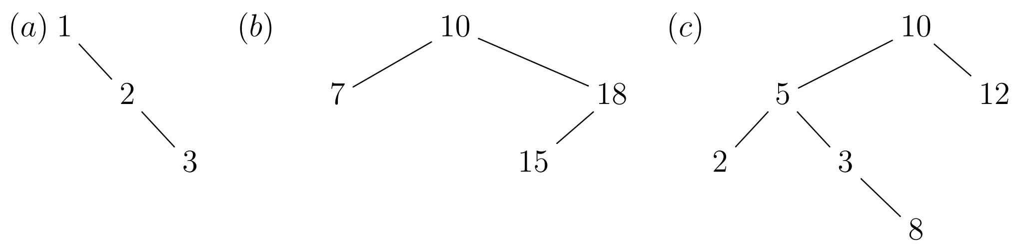
```
public static <E> Iterator<E>
find_first_equal(Iterator<E> begin, Iterator<E> end, E x) {
    Iterator<E> i = begin.clone();
    for (; !i.equals(end) && !i.get().equals(x); i.advance()) { }
    return i;
}
```

Rubric:

- Use of `clone` to create temporary iterator. (1 point)
- Use of iterator `equals` method to check if iterator has reached the `end`. (1 point)
- Use of iterator `get` to access the current element. (1 point)
- Use of `advance` to move the iterator. (1 point)
- Correct algorithm logic and return value. (6 points)

(It's OK to use a `while` loop instead of a `for` loop.)

8. [12 points] Which of the following trees are binary search trees? Which of them are AVL trees?



Solution:

(a) is a BST but not an AVL tree. (4 points)

(b) is a BST and an AVL tree. (4 points)

(c) is not a BST and is not an AVL tree. (4 points)

9. [8 points] Show that $3n + n \log_2 n \in O(n^2)$ using the definition of big-O.

Solution:

By the definition of big-O, we need to show that **(2 points)**

$$\exists kc. \forall n \geq k.\ 3n + n \log_2 n \leq cn^2$$

Choose $c = 1$ **(3 points)**.

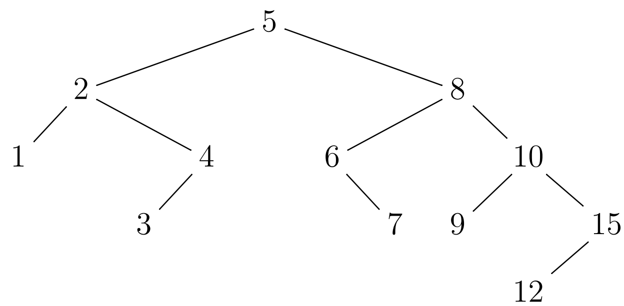
Towards finding a value for k, we compute a table for a few values of n:

n	$\log_2 n$	$n \log_2 n$	$3n$	$3n + n \log_2 n$	n^2
1	0	0	3	3	1
2	1	2	6	8	4
4	2	8	12	20	16
8	3	24	24	48	64
16	4	64	48	112	256
64	6	384	192	576	4096

Choose $k = 8$ **(3 points)**.

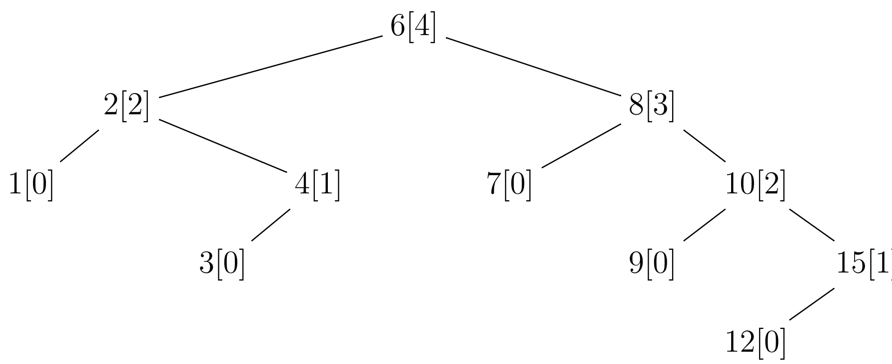
(Note: there are many other choices for c and k that are also correct, such as $c = 2, k = 2$.)

10. 8 points Given the following AVL binary search tree, remove key 5, maintaining the binary search tree and AVL properties. Explain each change that you make to the tree and draw the tree after each change.

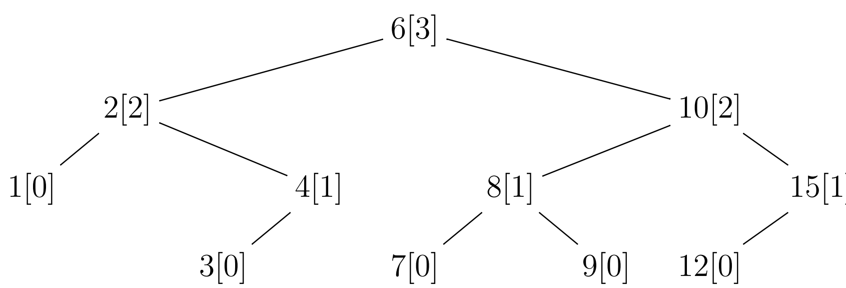


Solution:

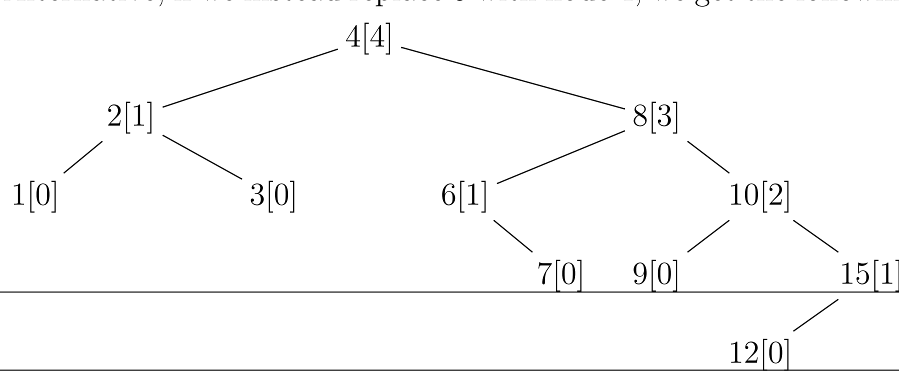
We first replace node 5 with node 6 **(3 points)**. (Alternatively, one could replace 5 with node 4, see below.) (The height of each node is written inside the brackets.)



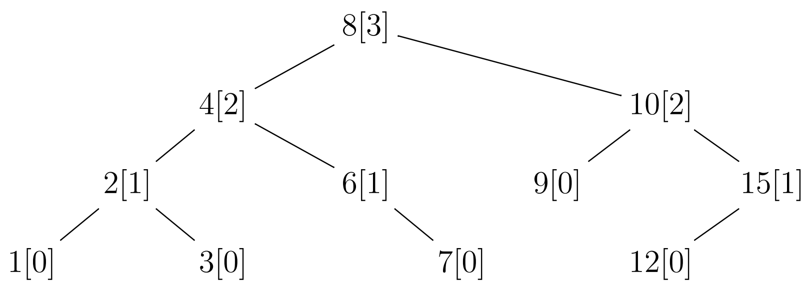
Node 8 is not AVL **(2 points)**, so we rotate 8 to the left **(3 points)**.



Alternative, if we instead replace 5 with node 4, we get the following tree **(3 points)**.



Node 4 is not AVL **(2 points)** so we rotate 4 to the left **(3 points)**.



11. [9 points] What is the big-O time complexity of the following anagram detection function in terms of the sum n of the lengths of the two input strings? Provide an argument for your answer that analyzes each method and loop and how their individual time complexities combine into the total time complexity.

```
private static LinkedList<Character> copy_without_spaces(String s) {
    LinkedList<Character> c = new LinkedList<>();
    for (int i = 0; i != s.length(); ++i) {
        if (s.charAt(i) != ' ') {
            c.add(s.charAt(i));
        }
    }
    return c;
}
public static boolean find_remove(LinkedList<Character> l, Character c) {
    Iterator<Character> iter2 = l.iterator();
    while (iter2.hasNext())
        if (iter2.next() == c) {
            iter2.remove();
            return true;
        }
    return false;
}

public static boolean anagram(String s1, String s2) {
    LinkedList<Character> l1 = copy_without_spaces(s1);
    LinkedList<Character> l2 = copy_without_spaces(s2);
    for (Character c1 : l1) {
        if (! find_remove(l2, c1))
            return false;
    }
    return l2.size() == 0;
}
```

Solution: The time complexity of the `copy_without_spaces` is $O(n)$ because the body of the `for` loop is $O(1)$, it iterates $O(n)$ times, and multiplying yields $O(n)$. **(3 points)**

The time complexity of the `find_remove` function is $O(n)$ because the body of the `while` loop is $O(1)$, it iterates $O(n)$ times, and multiplying yields $O(n)$. **(3 points)**

The time complexity of the `anagram` function is $O(n^2)$ because the `for` loop iterates $O(n)$ times, its body calls `find_remove` which is $O(n)$, so multiplying the times produces $O(n^2)$. **(3 points)**